DEVELOPMENT OF AN ADA* PACKAGE LIBRARY

Dr. Bruce Burton and Mr. Michael Broido

Intermetrics, Inc.
Aerospace Systems Group
5312 Bolsa Ave
Huntington Beach, California 92649

ABSTRACT

A usable prototype Ada package library has been developed and is currently being evaluated for use in large software development efforts. The library system is comprised of an Ada-oriented design language used to facilitate the collection of reuse information, a relational data base to store reuse information, a set of reusable Ada components and tools, and a set of guidelines governing the system's use. The prototyping exercise is discussed and the lessons learned are presented. Our experiences in developing the prototype library and lessons learned from it have led to the definition of a comprehensive tool set to facilitate software reuse.

* Ada is a trademark of the U.S. Department of Defense (AJPO).

INTRODUCTION

With the rising demand for cost-effective production of software, software reuse has become increasingly important as a potential solution to low programmer productivity. In the Ada programming language, explicit support is provided for software reuse through the "package" and "generic" language features. Unfortunately, the concept of Ada software reuse is not a panacea for our current software productivity problems. The notion of software reuse has been popular for decades. But implementing high degrees of reuse has usually failed, with the exception of some efforts in fairly narrow areas (business and compiler applications). The challenge then, is to recognize the contributions that the Ada language can make to a software reuse effort while at the same time identifying and resolving

language-independent problems. Based on the promise of the Ada programming language we undertook the development of a prototype Ada package library.

The prototyping exercise included:

- an examination of the reasons for low software reuse in the past,
- identification of activities and tools which would support a reuse methodology that spans the software development life-cycle from requirements through maintenance,
- the development of a phased implementation plan for software reuse that defines a development path from prototype to an operational, multi-company, geographically distributed system,
- development of a prototype for that methodology,
- the development, acquisition, and evaluation of representative package entries, and
- an examination of user interface techniques that could be used to maximize communications between a reuse system and its users.

BACKGROUND

As discussed above, software reuse is not a new concept. Significant efforts have been underway since the early 1960's to improve software development productivity through reuse (consider the early observations of McIlroy about the benefits of reuse presented at the NATO Software Engineering meeting in Garmish in 1968) [STANDISH83]. An analysis of the problems attending reuse has led to the identification of several potential hindrances to reuse [STANDISH83, BROIDO85]. These impediments to reuse can be categorized as technical, economic, and political obstructions. Some typical problems that hinder reuse include:

- lack of universal standards for component composition, level of documentation, coding techniques, testing, etc.,
- difficulty in transferring an understanding of the purpose of a software routine from the author to the potential reuser,
- higher initial development costs and longer schedules,
- risk management issues such as warranty, liability, and accountability,

- the "not invented here" syndrome, and
- the lack of pride typically exhibited when reuse has been selected in a software development project over original development.

While the problems impeding reuse are significant, the large size and cost of a major software development effort provides substantial motivation to improve productivity through reuse. Although Ada provides a natural vehicle for encouraging software engineering reuse, the same technical and political obstructions that have limited reuse in the past are likely to once again impede the sharing of software engineering products across projects. The Software Technology Department within Intermetrics is actively investigating the problems that hinder reuse. We are determined to find solutions to these problems and to collect and reuse Ada packages.

APPROACH

Along these lines, we have defined a phased approach to the development of a reusable package library suitable for use on large Ada applications projects. Rather than define an elaborate reuse facility and implement the library in a single step, we are currently prototyping parts of this facility to investigate the potential

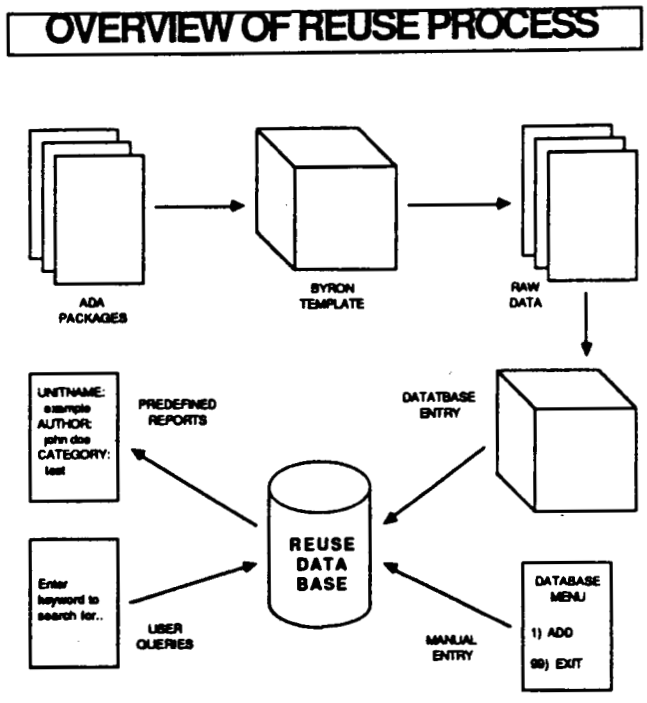

Figure 1. Reuse process overview.

utility of our approach. A complete description of this phased development plan is offered in [BURTON85]. The initial effort on this project has been focused on the creation of an Ada Software CATalog (ASCAT).

An overview of the ASCAT portion of the Ada package reuse system is shown in Figure 1. The system has been implemented using Byron*, Intermetrics' Ada-based program design language, and a commercial relational database management system. Central to the system is the ability of Byron to support definition and use of user-defined keywords.

Software Classification and Data Element Selection

One key to the success of

any reuse scheme is the types of classifications assigned to entries. The primary purpose of these classifications is to facilitate retrieval, but they may also be used to assist in defining storage strategies as well.

Selecting the classifications to be used is really a subset of a larger question: what data elements do we want to be able to retrieve about a particular entry? The list of storable elements seems in our opinion to be highly influenced by the size of the library (number of program units stored) and the degree of cooperation (or potential antagonism) among the users of the library. An initial cut at such a list was prepared [BROIDO85] from the perspective of our ultimate (multiple sites, multiple organizations, multiple usage types) system. Over 60 items which could potentially affect the suitability of an entry were named in seven major categories: identification (3 items), description (16 items), component parts (20), environment/usage (9), ordering information (7), and revision history (11). Even at this length, we recognize that there are undoubtedly many other items which could be added.

This list was far too large for our prototype, so we examined the context in which the prototype would operate. We characterized our initial environment as follows:

- All the users would be from the same company, although there would be several divisions using the common library. Thus, no restrictions on access would need to be supported.
- All initial entries would be written (when possible) in machine-independent Ada, so the compilation and execution environments would be well-defined.
- Source code would always be available, so users could do their own tailoring (no "black boxes"). Support in the form of corrections and training (other than by reading the source code) would not be provided.
- Emphasis was centered around the collection of reusable Ada packages rather than complete programs. Two factors influenced this decision. The first is that most of the packages we wanted to include already existed prior to the start of our efforts, and coherent design documents were not always available. The second factor was the widely distinct set of users we were addressing; they do not share the commonality of purpose which makes

---

* Byron is a trademark of Intermetrics, Inc.

domain analysis an effective top-down approach. The decision to center our design on packages enabled us to define a standard header for each package, based on the requirements of our Byron program product. Formalized requirements and design documentation were not required.

This decision causes the library to be more supportive of "bottom up" software construction techniques than most of today's top-down methods. The top-down methods reflect an attitude of defining what would be a perfect system and do not adequately recognize the influence of existing tools (including code) should have on requirements formulation in the presence of real cost constraints. (Note that the "object oriented design" strategies that are emerging with Ada reflect a tendency away from strict top-down methods.)

- No a priori naming conventions were established, although an informal guideline was prompted by the technical monitor of one of the contributing programs.

- Configuration management was not rigidly enforced, except within the rules imposed by Ada. In particular, no computerized list of outstanding users (people or programs) of the library routines was maintained.

- The programs which were intending to take advantage of the library provided no explicit funding for tool support or to ensure that any new packages created were generalized and otherwise suitable for future reuse. Package headers and other programmer-supplied information had to be easy (in both time and difficulty) for the programmers to supply.

- Various standards were established for the data items we would collect. Since we were attempting to catalog packages which had been previously created to support several different projects, It was necessary to retrofit many of the selected packages to include the required Byron comments. Part of our evaluation will be to try to identify the difficulties caused by "loose" definitions of essentially narrative fields (e.g., overviews). In addition, no common methodology had been established, so the degree of formality and the list of available support items (repeatable test cases, previous

sample output, user documentation, etc.) also varied considerably.

We filtered the original list down to the following data items for the database (others, such as the calling conventions and parameters, would be available from the source code if not given in the overview):

1. Unit name
2. Author
3. Unit size
4. Source language
5. Date created
6. Date last updated
7. Category code (see below)
8. Overview
9. Algorithm description
10. Errors/exceptions generated
11. Up to 5 keywords (for retrieval)
12. Machine dependencies (if any)
13. Program dependencies (if any)
14. Notes

Our retrieval strategy was based upon a combination of two alternate mechanisms. The first mechanism was the assignment of a hierarchical category code, with the hierarchy defined ahead of time and changeable only at well separated time intervals. This scheme is similar in concept to the ones used by Computing Reviews [ACM85] and the IMSL library [IMSL76]. But it was necessary to invent our own classification scheme since neither of those two was suitable to our purposes. Our scheme has the advantage that everyone knows what the codes are and can use an effectively finite procedure for searching the entries. Disadvantages include a growing list of vastly dissimilar "miscellaneous" entries and the inability of the original hierarchy designers to provide sufficiently discriminatory categories to provide effective retrieval (not too many or too few candidates).

For the second mechanism, we allowed the submitters to supply up to five keywords to be associated with each package. These keywords are not associated (as implicitly occurs within the hierarchy of categories), allow for overlapping topics (the packages do not conveniently fall into strict tree classifications), and can grow (without reprogramming or an all-knowing database administrator) with the needs of the projects they are created for. A scheme similar to this has been employed on NASA's COSMIC (Computer Software Management Information Center) system on complete programs, although the keywords allowed are suggested by the program authors and filtered by an acceptance team.

One of the authors is a member of the Applications Panel of the Department of Defense's Software Technology for Adaptable, Reliable Systems (STARS) Program. An important open issue surrounding the formation of a

potential Ada package library to be available as GFE materials for DoD contracts is defining the quality of the entries. On the one hand, some people advocate including only items of the highest quality, with full DoD standard documentation and even formal independent validation and verification (IV&V) required on new entries. Others prefer to let a more flexible scheme apply, with a "trust level" associated with entries. This latter scheme encourages "promotion" of existing entries from "buyer beware" to higher trust levels; after all, using informally qualified designs and code and then adding formal testing and documentation can still take less time (and often risk) than inventing from scratch. For the prototype, we decided to let all submitted entries be accepted and then evaluate the impact of this decision.

Reuse Information Extraction Mechanism

Another critical phase in the development of an Ada package library involves the extraction mechanism used to collect reuse-oriented information. The extraction mechanism utilized in an Ada package library must eventually provide several different capabilities to insure efficient operation. These required capabilities include:

- support for automatic data collection,
- support for insuring standardization of data entries,
- support for assuring continuity and consistency of reuse information across the Software Development Life Cycle (SDLC),
- support for checking completeness and reasonableness (e.g., dates), and
- support for reuse information examination.

The reuse information extraction approach utilized in our Ada package library is detailed in Figure 2. An analysis of this figure reveals that each of the elements previously identified for data collection has been mapped into predefined or user-defined keywords for the Byron design tool. A Byron template program was subsequently developed to automatically extract the reuse-oriented information. This information is placed into a file that can be directly processed into the ASCAT data base.

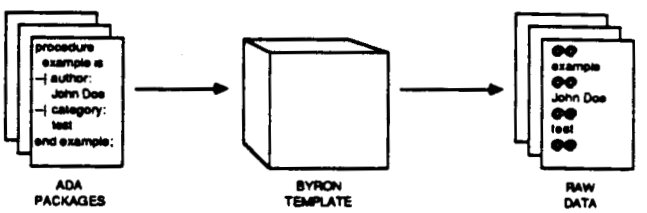

Figure 2. Extraction mechanism overview

The use of a Byron-oriented reuse information extraction mechanism provides most of the required capabilities enumerated above. This approach provides a means for automatic collection of data standardized in field name and format. Since the Byron design file is intended to transition into the implementation with reuse data intact, support is offered to assure information continuity across multiple phases of the SDLC.

While this extraction approach has many positive features, it is not without its shortcomings. The lack of predefined reuse attributes within Byron fails to support direct examination of reuse data items for completeness, consistency, and reasonableness. The inclusion of reuse-oriented information into the Byron-produced program library represents a simple potential improvement to our approach that could aid in the examination of the reuse data items.

Software Catalog Implementation

The software catalog for the reusable package library was implemented through the use of a commercial relational data base management package. The data definition capability used for field definition and the built-in data base programming language facilitated the examination of reuse data for limited correctness and consistency checking. The use of a data base also aided in the rapid development of an interface between the software catalog and potential Ada package users through the utilization of predefined reports and support for ad hoc user queries. Nonetheless, the user interface represents a weak link in our prototype package library. The present interface is very limited in the sense that it offers no context-specific support for communication between the reuse system and its users.

The present software catalog is limited in its interaction with the user. For example, consider the scenario of a software engineer performing an application software design of a routine that requires a sorting package. In the present system, the software engineer would need to: 1) exit the editor, 2) enter the software catalog data base system, 3) enter a query to identify the available sorting packages, 4) select the desired package, and 5) re-enter the editor and issue the necessary commands to draw the desired package (design/-code) into the applications program design.

This initial prototype software catalog can readily be improved to enhance the way in which it interacts with user. In Figure 3, the present mode of interaction is depicted. In Figure 4, another potential scenario is shown. In this scenario, a multi-window environment

is used where the user may perform the software catalog inquiry and concurrently examine several promising packages without exiting the editor.

A third possible operational scenario of the software catalog is not pictured. In this third approach, the data base query language would be replaced by a natural language front-end, the software catalog search would be assisted by an expert system, and the multi-window

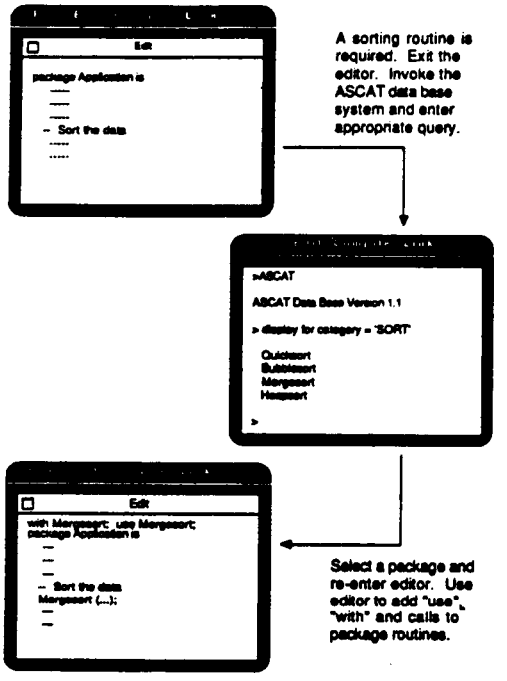

Figure 3. Current ASCAT operational scenario

approach would be supported by a language- and context-sensitive editor. The third approach is feasible with investigation into its implementation occurring in several current projects [ANDERSON85].

Intermetrics is currently investigating the implementation of this third approach. We are integrating a commercial natural language language front-end on our reuse database and are designing an expert system to facilitate evaluation and selection of alternative Ada packages. Although it is premature for significant conclusions on our expert system efforts, we have made several observations about the advantages and disadvantages of the Natural Language Front-End (NLFE).

Our preliminary findings on the NLFE are not surprising. As expected, we found the NLFE to be significantly easier to use than the traditional database query language supplied with our DBMS. On the negative side, we found that the natural language interface was substantially slower than our traditional database query language. Our initial figures show a performance penalty associated with the NLFE which ranged from a factor of five for relatively simple queries to a factor of ten for fairly complex queries.

Our preliminary query composition comparisons and initial performance evaluations show that the NLFE approach is a viable alternative to traditional database query languages. We are currently addressing the performance issues that plague the NLFE

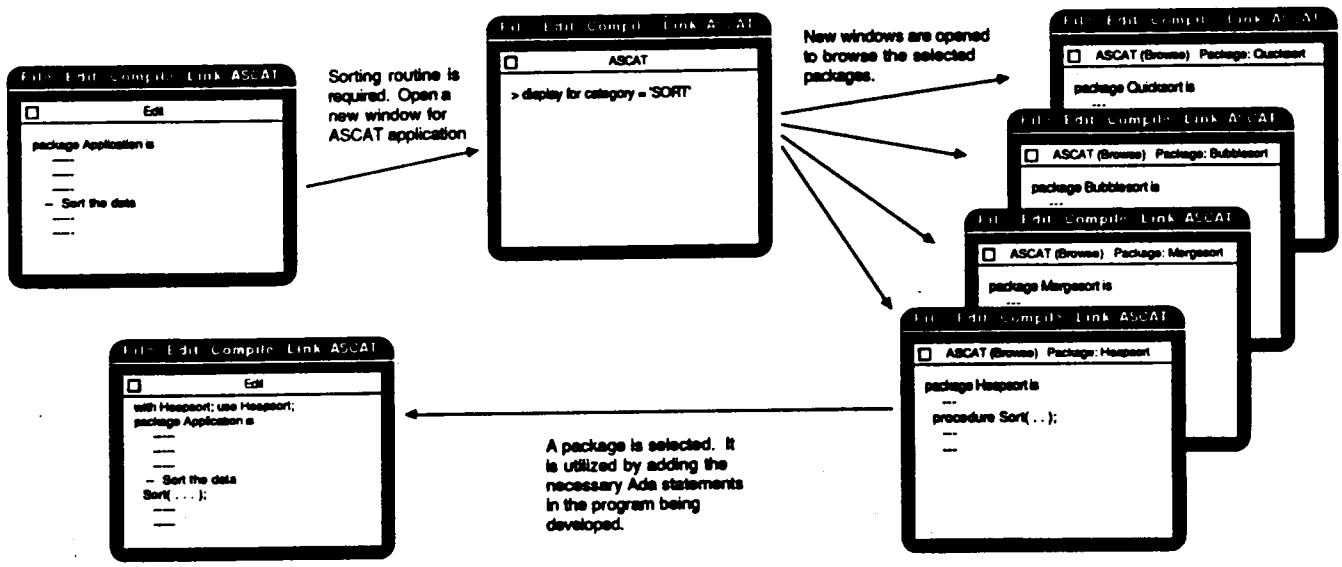

Figure 4. Improved ASCAT operational scenario

approach. We feel that the application of NLFE and expert system technology to the software library area will significantly simplify the operation of a software library and substantially improve the productivity of the software library users.

LESSONS LEARNED

The development, collection, evaluation, and cataloging of reusable components and tools undertaken in the development of an Ada package library has led to some interesting observations concerning Ada package reuse. Unfortunately, we do not yet have enough experience to evaluate the selected category scheme, keyword retrieval capability, or the list of collected data elements.

During the past year, we have developed a set of test and analysis tools written in Ada and intended for Ada software development efforts. The fixed-price nature of this contract and the fact that it represented the first major Ada development contract within our division motivated us to emphasize reuse of existing Ada packages as a cost and risk reduction measure. Based on the results of that contract, we found that reuse of existing generic support packages significantly improved our productivity, with over 33% of the code comprised of reused packages.

On the negative side, we found that several of the tools initially exhibited poor performance. In almost every instance, we found the general nature of the reused packages to contribute heavily to the performance problems. We also found that the generic Ada packages offered much more functionality than required in our application. The extra functionality resulted in a size penalty with respect to the executable code. The use of a performance analyzer and tailoring of the reused code for the current application substantially improved tool performance [RATHGEBER86].

We also studied the problem of composing reusable applications packages from existing reusable components. As part of an Air Force study, we compared the performance of two different implementations of reusable Kalman filter routines. One of the routines was written in Ada; generic Ada mathematics packages were heavily used in its development. The other routine was written in FORTRAN and specifically designed to solve a specific Kalman filter problem. A performance comparison of the generalized Ada package against the custom-tailored FORTRAN routines showed the FORTRAN routine to exhibit significant speed advantages over its Ada counterpart. This performance difference is probably due to the relative immaturity of the Ada compiler used in this study and also to the generalized nature of the Ada packages. An important conclusion of the study is that the performance problems associated with including a generalized reusable Ada package into an applications program are substantially compounded when an entire system is comprised of reusable components which also consist of reusable components.

Although many of our lessons learned have negative implications for the use of Ada reusable packages, there is some light at the end of the tunnel. Reuse was a big aid in increasing our productivity in the development of Ada test and analysis tools. We also found that reuse can be successfully employed in the development of efficient Ada systems if sufficient thought is put into how the packages are to be reused and if the proper tools are available (e.g., such as a performance analyzer).

CONCLUSIONS

In accordance with our previous plan, we have completed a prototype mechanism for extracting reuse information from packages developed in the normal course of business. We also have a primitive mechanism for entering that data in a catalog and searching the catalog for entries that are potentially useful on new projects. The approach centers on the design and implementation phases, since these are the ones to which

reuse concepts may most readily be applied in the given environments.

We have confirmed with actual experience our earlier assessment that successful implementation of a reuse methodology requires thought, action and management direction and support throughout the software life cycle. This, however, may require a management reorientation to the view of software development as the acquisition of a long-lived corporate asset rather than as only the work required to produce the current deliverable [WEGNER84, YEH85]. Complementing the reuse efforts being conducted by the STARS office, which are targeted at long range objectives, our approach provides useful tools which can be utilized immediately.

We have achieved some success in applying software reuse. Effective use of the packages forced us to define subsets of them which subsequently required performance tuning. This points out the value of developing a comprehensive reuse methodology, with adequate support tools to facilitate the development of efficient systems comprised of reusable components.

The Ada language and the methodologies growing up around it provide a good start toward achieving larger scale reuse than we have achieved in the past. But they are not enough by themselves. Even with Ada, there are still plenty of obstacles to reuse. A management commitment and desire to improve productivity when coupled with a comprehensive reuse methodology and the proper tools offer substantial promise for improvement.

REFERENCES

ACM85 "Introduction to the CR Classification System," Computing Reviews, Vol. 26, No.1. Association for Computing Machinery, January, 1985, pp. 45-57.

ANDERSON85 Anderson, C.M. and McNicholl, D.G., "Reusable Software - A Mission Critical Case Study", AIAA Computer in Aerospace V Conference, October 21-23, Long Beach, California.

BROIDO85 Broido, Michael D., "Software Commonality Study for Space Station Phase B", Intermetrics Report IR-CA-029, Intermetrics, Inc., 29 May 1985.

BURTON85 Burton, B.A. and Broido, M.D., "A Phased Approach to Ada Package Reuse", STARS Workshop on Software Reuse, April 9-12 1985, Naval Research Laboratory, Washington, DC 20375-5000.

IMSL76 Reference manual, The International Mathematical & Statistics Libraries, IMSL, Fall, 1976.

RATHGEBER86 Rathgeber, R.L., "Technical Report on Ada Test and Analysis Tools", Intermetrics, Inc., Huntington Beach, California, In Preparation.

STANDISH83 Standish, T.A., "Software Reuse", presented at the ITT Workshop on Reusability in Programming, Rhode Island, September 7-9, 1983.

WEGNER85 Wegner, Peter, "Capital- Intensive Software Technology," IEEE Software, Vol. 1, No.3, IEEE Computer Society, July, 1984, pp. 7-45.

YEH85 Yeh, Dr. Raymond T., "Japanese and Brazilian Software Technology Initiatives". (Luncheon address). Published by the NSIA Software Committee in the Proceedings of the First DOD/Industry STARS Program Conference, 30 April 1985 - 2 May 1985.